

L3 informatique

2022-2023

Programmation Web

Partie 4 : JavaScript

Étienne André

Université Sorbonne Paris Nord
Etienne.Andre@univ-paris13.fr



Version: 8 février 2023

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements
- 8 Dessiner en JavaScript

JavaScript

- Langage de programmation de scripts
- Créé en 1995
- Surtout utilisé dans les pages Web
 - interprété par le navigateur Web
 - donc une exécution côté client

JavaScript

- Langage de programmation de scripts
- Créé en 1995
- Surtout utilisé dans les pages Web
 - interprété par le navigateur Web
 - donc une exécution côté client



Pas de rapport avec Java

JavaScript n'a (aujourd'hui) aucun rapport avec le langage Java

Versions de JavaScript

- Deux versions principales cohabitent : ES5 et ES6
- ES6 **ajoute** des constructions à ES5 mais ne les remplace pas
- Le passage de ES5 à ES6 se fait progressivement au fil des versions des navigateurs
- Presque tout ES6 est implanté dans les navigateurs récents, mais pas les anciens

Principales caractéristiques de JavaScript

- Langage objet à prototype

- Les objets ne sont pas (forcément) des instances de classes
- Les fonctions sont des objets de première classe (utilisables sans restrictions, y compris comme paramètres)

Principales caractéristiques de JavaScript

■ Langage objet à prototype

- Les objets ne sont pas (forcément) des instances de classes
- Les fonctions sont des objets de première classe (utilisables sans restrictions, y compris comme paramètres)

■ Langage impératif (et fonctionnel!)

- autres langages impératifs : C, Java, C++, Python...
- autres langages fonctionnels : Lisp, Haskell, OCaml...

Principales caractéristiques de JavaScript : typage

- Langage **faiblement typé**

- autorisé :

```
1 console.log("20" + 46); // affiche "2046"
```

- autres exemples : C, C++

- contre-exemples (fortement typés) : Java, OCaml, Python

Principales caractéristiques de JavaScript : typage

■ Langage faiblement typé

- autorisé :

```
1 console.log("20" + 46); // affiche "2046"
```

- autres exemples : C, C++
- contre-exemples (fortement typés) : Java, OCaml, Python

■ Langage à typage dynamique

- autorisé :

```
1 let ma_var = 2046;  
2 ma_var = 'ITMFL'; // pas d'erreur de typage
```

- autres exemples : Perl, PHP, Python...
- contre-exemples (typage statique) : Ada, Java, C, C++...

Contexte de ce cours

Contours du cours

Ce cours s'intéresse à l'utilisation de JavaScript dans un contexte de programmation Web, aux côtés de HTML et CSS.

Outline

- 1 Définition et syntaxe de base
 - **Syntaxe de base**
 - Les variables
 - Les valeurs
 - Les structures de contrôle
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements

Bref aperçu de la syntaxe

```
1 // Commentaire
2 let x;           // déclaration de variable
3 let y = 2046;    // déclaration + initialisation de variable
4 var z = 'ITMFL'; // ancienne notation (ES5)
5 x = y + 1;      // affectation
6
7 foo(x, y);      // appel de fonction avec args x,y
8 objet.bar(z);   // appel de la méthode 'bar' de 'objet'
9
10 // conditionnelle
11 if(x === 0){    // x est-elle égale à 0 ?
12     x += 42;
13 }
14
15 // itérations
16 while (x > 0){
17     x = x-1;
18 }
19 for (let i = 0; i<10; i++){
20     console.log(i);
21 }
```

Mode strict

- JavaScript possède 2 modes de fonctionnement

- 1 le mode **normal** ou **sloppy**

- 2 le mode **strict**

- Le mode strict peut être activé :

- pour un fichier complet avec en début le tag suivant :

```
1 'use strict'
```

- par fonction :

```
1 function fonctionEnModeStrict () {  
2     'use strict';  
3     ...  
4 }
```

- Le mode strict génère plus de *warnings* et d'erreurs mais est souvent **plus rapide**

Outline

- 1 Définition et syntaxe de base
 - Syntaxe de base
 - **Les variables**
 - Les valeurs
 - Les structures de contrôle
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements

Variables

- Doivent être **déclarées** (en mode strict)

```
1 let v;
```

- Peuvent être **initialisées** à la déclaration

```
1 let v = 2046;
```

- Sont typées **dynamiquement** (pas de typage à la déclaration)

```
1 let v = 2046;  
2 v = "WKW";  
3 v = false;
```

- Syntaxe des noms de variable : $(\text{lettre} \mid \$ \mid _)(\text{lettre} \mid \text{chiffre} \mid \$ \mid _)^*$

```
1 // Exemples de déclarations de variables  
2 let i, $v, _x, _12m, $2046, THX1138, _2001;
```

Déclaration et portée des variables

Deux syntaxes de déclarations de variables

- Mot-clé **var** (ES5) : la portée (*scope*) est l'ensemble de la fonction où elle est déclarée
- Mot-clé **let** (ES6) : la portée (*scope*) est limitée au bloc où elle est déclarée
- Une variable déclarée en dehors de tout bloc est **globale** (à éviter!)

```
1 function f(){  
2   var v = 2046;  
3   if (v < 3000){  
4     let w = 3000-2046;  
5     var z = w+v;  
6     console.log(v, w); // OK  
7   }  
8   console.log(v,z,w); // not OK  
9 }
```


Déclaration et portée des variables

Deux syntaxes de déclarations de variables

- Mot-clé **var** (ES5) : la portée (*scope*) est l'ensemble de la fonction où elle est déclarée
- Mot-clé **let** (ES6) : la portée (*scope*) est limitée au bloc où elle est déclarée
- Une variable déclarée en dehors de tout bloc est **globale** (à éviter!)

```
1 function f(){
2   var v = 2046;
3   if (v < 3000){
4     let w = 3000-2046;
5     var z = w+v;
6     console.log(v, w); // OK
7   }
8   console.log(v,z,w); // not OK
9 }
```

Uncaught ReferenceError: w is not defined

Outline

- 1 Définition et syntaxe de base
 - Syntaxe de base
 - Les variables
 - **Les valeurs**
 - Les structures de contrôle
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements

Les valeurs

■ Valeurs primitives

- booléens: `true`, `false`
- nombres: `2046`, `3.141592653`
- chaînes: `'Quentin'`, `"Tarantino"`
- `undefined`, `null`

■ Objets

- Objets classiques: `{ prenom : 'Quentin', nom : 'Tarantino' }`
- Tableaux: `[2, 0, 4, 6]`
- Fonctions et expressions régulières

■ Les valeurs ont un type

- Voir l'opérateur `typeof`

```
> typeof 2046
"number"
> typeof {}
"object"
> typeof undefined
"undefined"
```

Les valeurs primitives

- Valeurs primitives : objets non modifiables
 - Possèdent des propriétés et méthodes
 - Ne peuvent pas être changées
- Comparaison par valeur

```
> 3 === 3
true
> let s = 'WKW'
> s === 'WKW'
true
> s.length
3
> s.length = 5
5
> s.length
3
```

```
# attention :
  transtypage
> 2046 === "2046"
false
> 2046 == "2046"
true
# éviter d'utiliser
  ==
```

Les nombres

- Les nombres sont tous des flottants
- Opérateurs : `+`, `-`, `/`, `*`, `++`, `--`, `%`
- Deux valeurs spéciales : `Infinity`, `NaN` (*not a number*)

```
> 1 === 1.0
true
> 3 / 0
Infinity
> Number('WKW');
NaN
> let i = 6;
> 5 + --i
10
```

Les booléens

- Deux valeurs : `true`, `false`
- Valeurs fausses : `undefined`, `null`, `false`, `0`, `NaN`, `''`
- Opérateurs à résultat booléen :
 - Comparaisons : `<`, `<=`, `>`, `>=`, `===`, `!==` (ainsi que `==`, `!=` : à éviter)
 - Connecteurs booléens : `!`, `&&`, `||` (opérateurs court-circuit)

```
# illustration du court-circuit : foo() n'est jamais
  appelée ci-dessous :
> false && foo()
false
> true || foo()
true
```

Les chaînes de caractères

■ Deux syntaxes de guillemets : 'simples' ou "doubles"

```
1 let prenom = "Quentin";  
2 let nom    = 'Tarantino';
```

■ Concaténation

```
1 console.log(prenom + " " + nom);  
2 // affiche "Quentin Tarantino"
```

■ Propriétés

```
1 console.log(prenom.length); // affiche 7
```

■ Méthodes, par ex :

```
1 console.log(nom.toUpperCase()); // affiche "TARANTINO"
```

Les templates de chaînes

- *String template* : Expression de chaîne de caractères permettant d'inclure des variables et des expressions évaluées au moment de l'affectation
 - Depuis ES6

```
1 let nom      = 'Sciamma';
2 let prenom   = 'Céline';
3 let naissance = 1978;
4 let annee_courante = new Date().getFullYear();
5 let phrase = `La réalisatrice ${prenom + ' ' + nom} a
6             cette année ${annee_courante - naissance} ans`;
7 console.log(phrase);
// affiche "La réalisatrice Céline Sciamma a cette
//         année 45 ans"
```


Outline

- 1 Définition et syntaxe de base
 - Syntaxe de base
 - Les variables
 - Les valeurs
 - **Les structures de contrôle**
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements

Les structures de contrôle

Les structures de contrôle en JavaScript sont similaires à celles rencontrées dans de nombreux langages impératifs

- `if (...) then {...} else {...}`
- boucle `while(){...}` (et variante `do { ... } while(...)`)
- boucle `for(){...}`
- boucle `for (... in...){...}`

Les structures de contrôle : switch

- Objectif : effectuer un traitement particulier sur une valeur en énumérant des cas possibles

```
1 let nourriture = window.prompt("Que voulez-vous manger
  ?", "asperges");
2 switch (nourriture) {
3   case 'asperges':
4   case 'aubergines':
5     console.log("Vous aimez les légumes");
6     break;
7   case 'oranges':
8   case 'pommes':
9     console.log("Vous aimez les fruits");
10    break;
11   case 'kouign-amann':
12     console.log("Vous aimez le beurre");
13     break;
14   default:
15     console.log("Vous aimez des choses bizarres");
16 }
```

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions**
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements
- 8 Dessiner en JavaScript

Les fonctions en JavaScript

- Une fonction JavaScript est un **objet**; par conséquent, elle
 - peut être **créée** via un littéral
 - peut être **affectée** à des variables, tableaux, propriétés
 - peut être **passée en paramètre** à d'autres fonctions
 - peut être **retournée comme résultat** d'une autre fonction
 - possède des **propriétés** et des **méthodes**
 - peut être **invoquée** en plaçant des parenthèses :

```
1 ma_fonction();
```

Types des fonctions

Plusieurs types de fonctions :

- 1 Les fonctions **natives** appartenant au langage JavaScript
 - Exemples : `parseInt()`, `isNaN()`, `eval()`...
- 2 Les fonctions (méthodes) relatives à des objets de JavaScript
 - Exemples : `Math.random()`, `Math.ceil()`, `window.alert()`, `window.prompt()`, `document.write()`...
- 3 Les fonctions définies dans le script

```
1 function age(naissance){  
2   let annee_courante = new Date().getFullYear();  
3   return annee_courante - naissance;  
4 }
```

Fonctions natives

1 Conversions de types :

- `parseInt()` : conversion d'une chaîne de caractères en entiers
- `parseFloat()` : conversion d'une chaîne de caractères en réels (renvoie `NaN` si ce n'est pas un nombre)

2 Détection de valeurs exceptionnelles :

- `isNaN()` : indique si l'argument est `NaN`
- `isFinite()` : indique si l'argument est une valeur numérique finie

3 Code d'évaluation : fonction `eval()`

- analyse une chaîne et l'exécute comme s'il s'agissait d'un script JavaScript

Fonctions natives

1 Conversions de types :

- `parseInt()` : conversion d'une chaîne de caractères en entiers
- `parseFloat()` : conversion d'une chaîne de caractères en réels (renvoie `NaN` si ce n'est pas un nombre)

2 Détection de valeurs exceptionnelles :

- `isNaN()` : indique si l'argument est `NaN`
- `isFinite()` : indique si l'argument est une valeur numérique finie

3 Code d'évaluation : fonction `eval()`

- analyse une chaîne et l'exécute comme s'il s'agissait d'un script JavaScript

Fonction `eval`

La fonction `eval` de JavaScript peut être dangereuse : possibilité d'exécuter du code JavaScript, potentiellement néfaste.

Fonctions relatives à des objets JavaScript

■ Exemple : l'objet `window`

```
1 window.alert("Message à afficher");
```

Affiche une boîte de dialogue avec la chaîne de caractères en paramètre

```
1 let resultat = window.prompt("Message affiché",  
    "Valeur par défaut affichée");
```

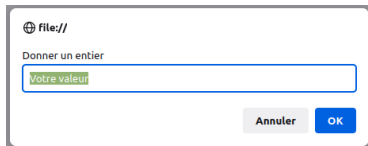
Affiche une boîte de dialogue attendant une réponse de l'utilisateur ou utilisatrice qui sera stockée dans la chaîne de caractères `resultat`

Fonctions natives : exemple

```
1 let nombreChaine = window.prompt("Donner un entier", "Votre  
   valeur");  
2 let nombre = parseInt(nombreChaine);  
3 if(nombre % 2 === 0){  
4     document.write('<p>' + nombre + ' : entier saisi  
   pair</p>');  
5 }else{  
6     document.write('<p>' + nombre + ' : entier saisi  
   impair</p>');  
7 }
```

Fonctions natives : exemple

```
1 let nombreChaine = window.prompt("Donner un entier", "Votre  
   valeur");  
2 let nombre = parseInt(nombreChaine);  
3 if(nombre % 2 === 0){  
4     document.write('<p>' + nombre + ' : entier saisi  
   pair</p>');  
5 }else{  
6     document.write('<p>' + nombre + ' : entier saisi  
   impair</p>');  
7 }
```



A screenshot of a browser's prompt dialog box. The title bar shows a globe icon and the text "file://". The main content area contains the text "Donner un entier" followed by a text input field with the value "votre valeur" and a green selection highlight. At the bottom right, there are two buttons: "Annuler" (grey) and "OK" (blue).

La déclaration de fonctions

- le mot-clé `function`
- un nom
- une liste (possiblement vide) de paramètres entre `(...)`
- une série (possiblement vide) d'instructions entre `{ ... }`

```
1 // déclaration d'une fonction de somme
2 function somme (a, b) {
3     return a+b ;
4 }
5 console.log(somme.name);           // affiche "somme"
6 console.log(somme(2000, 46));     // affiche "2046"
7
8 console.log(somme);
9 // affiche:
10 // fonction somme (a, b) {
11 //     length: 2
12 //     name: "somme"
13 //     prototype: Object { ... }
14 //     ...
```

Expressions de fonctions et fonctions anonymes

- Il est possible d'affecter ou passer en paramètre des **expressions de fonction** sans avoir à déclarer la fonction
 - ES5 : fonctions **anonymes** (elles n'ont pas de nom)
 - ES6 : le nom est **inféré**

```
1 let soustraction = function(a, b) {  
2   return a-b;  
3 }  
4 console.log(soustraction.name); // affiche "soustraction"  
5 console.log(soustraction(2050, 4)); // affiche 2046
```

Expressions de fonctions avec =>

- Depuis ES6 : nouvelle notation pour les expressions de fonctions avec =>

```
1 let produit = (a,b) => {return a*b;}
2 let wkw     = ()    => {return 2046;}
3 let incrBy1 = (x)  => x+1;
4 console.log(wkw);           // affiche "function
   wkw(){...}"
5 console.log(wkw());        // affiche "2046"
6 console.log(wkw.name);     // affiche "wkw"
7 console.log(produit(31,66)); // affiche "2046"
```

Invocation immédiate

- Une fonction ou une expression de fonction peut être invoquée **immédiatement** à sa création

```
1 // f1, f2, f3 sont des nombres (pas des fonctions !)  
2 let f1 = (function return42() {return 42;}) ();  
3 let f2 = (function () {return 300;}) ();  
4 let f3 = ((x) => 31 * x) (66);  
5 // f4 est une fonction  
6 let f4 = ((x) => 31 * x); // pas d'invocation immédiate  
7 console.log(f1); // affiche "42"  
8 console.log(f2); // affiche "300"  
9 console.log(f3); // affiche "2046"  
10 console.log(f4(66)); // affiche "2046"
```

Arguments des fonctions

- Lors d'un appel, le nombre d'arguments peut être différent du nombre de paramètres
 - arguments manquants : valeur `undefined`
 - arguments supplémentaires : dans le pseudo-tableau `arguments`

```
1 function f(x, y) {  
2   console.log("x=" + x, "; y=" + y + "; args=");  
3   console.log(arguments);  
4 }  
5 f(1,2); // affiche : "x=1; y=2; args=[1,2]"  
6 f(1,2,3); // affiche : "x=1; y=2; args=[1,2,3]"  
7 f(1); // affiche : "x=1; y=undefined; args=[1]"
```


Arguments des fonctions : valeur par défaut

- Depuis ES6 : possibilité de définir une **valeur par défaut**
- Notation explicite pour les paramètres optionnels
 - syntaxe : `argument=valeur_par_defaut`
- Notation explicite pour les arguments supplémentaires dans un tableau
 - syntaxe : `...nom_du_tableau`

```
1 // Si l'argument "y" est absent, il prend pour valeur 2046
2 // Tous les arguments supplémentaires seront stockés dans
  le tableau les_autres
3 let g = (x, y=2046, ...les_autres) =>
4 {
5   console.log("x=" + x, "; y=" + y + "; les_autres=");
6   console.log(les_autres);
7 }
8 g();           // affiche : "x=undefined; y=2046; les_autres=[]"
9 g(1);         // affiche : "x=1; y=2046; les_autres=[]"
10 g(1,2);       // affiche : "x=1; y=2; les_autres=[]"
11 g(1,2,3);     // affiche : "x=1; y=2; les_autres=[3]"
```

Fonction en paramètre (*callback*)

- Rappel : Une fonction est un objet et donc peut être passée en paramètre

```
1 // Définitions équivalentes
2 function somme1(a,b) {return a+b;}
3 let somme2 = (a,b) => a+b;
4
5 // Définition d'une fonction prenant une fonction f en
6 // paramètre :
7 function calculer(a, b, f) {return f(a,b);}
8
9 // On appelle calculer en lui passant une fonction en
10 // argument
11 console.log(calculer(20, 22, somme1)); // affiche 42
12 console.log(calculer(2000, 46, somme2)); // affiche 2046
13 // On peut passer une expression comme argument
14 console.log(calculer(31, 66, (a,b) => a*b)); // affiche 2046
```

Fonction comme résultat

- Rappel : Une fonction est un objet et donc peut être retournée comme **résultat** d'une fonction

```
1 function incrementeur(x) {  
2   // Ici on retourne une fonction à un argument "a"  
3   return (a) => { return a+x;};  
4 }  
5  
6 let incr5 = incrementeur(5);  
7 // incr5 est une fonction qui incrémente de 5  
8  
9 console.log(incr5(7)); // affiche 12  
10 console.log(incr5(2041)); // affiche 2046  
11 console.log(incrementeur(2)(2044)); // affiche 2046
```

Portée des fonctions

- Une fonction ou une variable peut être déclarée :
 - à tout moment
 - dans n'importe quelle fonction
- Rappel sur la portée (*scope*) des variables
 - Mot-clé `var` (ES5) : ensemble de la fonction où elle est déclarée
 - Mot-clé `let` (ES6) : bloc où elle est déclarée
- Portée (*scope*) des fonctions
 - tout le bloc de déclaration, ainsi que les fonctions et blocs imbriqués
 - les références en avant sont possibles

Portée des fonctions : exemple

```
1 function portee_fonctions() {
2   console.log('incrBy2(2046):', incrBy2(2046)); // OK
3
4   function incrBy2(x) {return x+2 ;}
5   if(true){
6     console.log('a:', a); // ReferenceError
7     console.log('b:', b); // affiche "undefined"
8     let a = 10;
9     console.log('incrByA(6):', incrByA(6));
10    function incrByA(x) {
11      console.log('incrBy2(100):', incrBy2(100));
12      return x + a;
13    }
14    var b = 8;
15    console.log('incrByA(3):', incrByA(3));
16  }
17  console.log('a:', a); // ReferenceError
18  console.log('b:', b);
19  console.log(incrByA(5)) // ReferenceError
20  console.log('incrBy2(40):', incrBy2(40));
21 }
```

Clôture (*closure*)

- Clôture (*closure*) d'une fonction : portée au sein de laquelle elle est déclarée
 - Inclut l'accès à tous les objets (variables) présents dans cette portée
- La clôture est **toujours accessible** lors de l'exécution de la fonction, même si la portée a disparu ou si la fonction est exécutée dans une portée différente!
- La fonction conserve une **référence** vers tous les objets présents dans sa portée initiale de déclaration

Clôture : exemple

- La portée de la fonction `f` consiste en les lignes 1 à 4
 - ce qui inclut la variable `i...` et l'argument `n`

```
1 function f(n) {  
2   let i=1;  
3   return (x) => (x + i++) * n;  
4 }  
5  
6 // f10 est une fonction  
7 let f10 = f(10);  
8  
9 // donc on peut invoquer f10:  
10 console.log(f10(5)); // affiche 60  
11 console.log(f10(5)); // affiche 70  
12 console.log(f10(5)); // affiche 80
```

- La valeur de `i` est modifiée à chaque appel de `f10`

Clôture : utilisations (1/2)

- Pour créer des **variables partagées** non globales

```
1 function sommeTab(tableau){
2   let somme = 0; // variable de somme
3   // f est une fonction qui incrémente somme de e
4   let f = (e) => {somme += e;};
5   tableau.forEach(f); // appel de f sur chaque élément
6   return somme;
7 }
8 console.log(sommeTab([1,2,3,4,5])); // affiche 15
```


Clôture : utilisations (2/2)

■ Pour « piéger » une variable

```
1 function sequence(u0){
2   let u = u0;
3   return () => u++;
4 }
5
6 let suivant1 = sequence(1);
7 let suivant10 = sequence(10);
8
9 console.log(suivant1()); // affiche 1
10 console.log(suivant10()); // affiche 10
11 console.log(suivant10()); // affiche 11
12 console.log(suivant1()); // affiche 2 (u est LOCAL à
   sequence1)
13 console.log(suivant10()); // affiche 12
```

Clôture : nœuds dans le cerveau

- Attention à la clôture (et donc à la portée des variables locales ou globales!)

```
1 // Déclarations
2 function incrementeurBizarre(valeur_init){
3     let u = valeur_init;
4     return () => {
5         pas++; v++; u += pas;
6         console.log("pas = " + pas + " ; v = " + v + " ; u = " + u);
7     }
8 }
9
10 let v = 0; // variable globale
11 if(true){
12     var pas = 0; // variable globale car 'var'
13 }
14
15 // Tests
16 let iB5    = incrementeurBizarre(5);
17 let iB100 = incrementeurBizarre(100);
18
19 iB5();    // affiche ?
20 iB5();    // affiche ?
21 iB100(); // affiche ?
22 iB100(); // affiche ?
23 iB5();    // affiche ?
```

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets**
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements
- 8 Dessiner en JavaScript

Objets : constructeur, propriétés, méthodes

■ Éventuel constructeur :

```
1 let maintenant = new Date();
```

■ Propriétés

```
1 monObjet.ma_propriete;
```

■ Méthodes

```
1 maintenant.getHours();
```

Exemple :

```
1 let maintenant = new Date();  
2 console.log("Il est " + maintenant.getHours() + "h" +  
   maintenant.getMinutes());  
3 // affiche par exemple "Il est 20h46"
```

Les objets en JavaScript

- Objets natifs

- Exemples : `Math`, `String`, `Array`, `Date`, `Number`...

- Objets liés aux pages Web

- Exemples : `document`, `navigator`, `window`...

- Objets créé par le script

Un exemple d'objet natif : l'objet `Math`

■ Quelques propriétés

```
1 Math.E; // le nombre d'Euler  
2 Math.Pi;
```

■ Quelques méthodes

- `floor` : Renvoie le plus grand nombre entier plus petit ou égal à un nombre
- `max` : Renvoie le maximum d'un ensemble de nombres
- `min` : Renvoie le minimum d'un ensemble de nombres
- `random` : Renvoie un nombre pseudo-aléatoire dans $[0, 1]$
- `round` : Renvoie la valeur d'un nombre arrondi à l'entier le plus proche.
- Et aussi : `abs`, `ceil`, `cos`, `sin`, `exp`, `sqrt`, `log`, `pow`, ...

Exemple :

```
1 console.log("Meilleure note arrondie par défaut : " +  
   Math.floor(Math.max(2.7, 17.4, 9.7))); // affiche "17"
```

Un exemple d'objet natif : l'objet `String`

- Exemple de propriété : `length`
 - Longueur en caractères
- Quelques méthodes
 - `charAt` : Renvoie le caractère à la position spécifiée
 - `substring` : Renvoie les caractères d'une chaîne entre deux positions dans celle-ci
 - `toLowerCase` : Renvoie la valeur de la chaîne appelante convertie en minuscules
 - `toUpperCase` : Renvoie la valeur de la chaîne appelante convertie en majuscules

Exemple :

```
1 let nom = prompt ("Votre nom ?");
2 // Je tape "André"
3
4 console.log("Nom en majuscules : " + nom.toUpperCase() + ",
   première lettre : " + nom.charAt(0) + ", longueur : " +
   nom.length); // Affiche "Nom en majuscules : ANDRÉ,
   première lettre : A, longueur : 5"
```

Créer ses propres objets

- ES5 : pas de classes... mais des objets!
- Ajout dynamique de propriétés à tout moment

```
1 var film = {
2   titre      : "Reservoir Dogs",
3   equipe_technique : {
4     montage   : "Sally Menke",
5     costumes  : "Betsy Heimann"
6   },
7   sortie     : 1992,
8   realisation : "Quentin Tarantino"
9 };
10 // Ajout dynamique d'une propriété
11 film.distinction = "Festival d'Avignon";
12 // Accès aux propriétés
13 console.log("Le film " + film.titre + " a été réalisé par "
  + film.realisation + ", monté par " +
  film.equipe_technique.montage + " et a reçu un prix au
  " + film.distinction); // affiche "Le film Reservoir
  Dogs a été réalisé par Quentin Tarantino, monté par
  Sally Menke et a reçu un prix au Festival d'Avignon"
```


Méthodes en ES5

- **Méthode** : propriété d'un objet dont la valeur est une fonction

```
1 var velo = {
2   marque   : "Giant",
3   bequille : true,
4   couleur  : "bleu",
5   vitesse  : 0,
6
7   demarrer : function() { this.bequille = false;},
8   accelerer : function(x) { if (!this.bequille)
9     this.vitesse +=x ; },
10  repeindre : function(c) { this.couleur = c;}
11 };
12 velo.demarrer();
13 console.log(velo.bequille) ; // affiche "false"
14 velo.accelerer(20);
15 console.log(velo.vitesse) ; // affiche "20"
```

Méthodes en ES6

■ Méthode : nouvelle notation dédiée

```
1 let velo = {
2   marque   : "Giant",
3   bequille : true,
4   couleur  : "bleu",
5   vitesse  : 0,
6
7   demarrer() { this.bequille = false;},
8   accelerer(x) { if (!this.bequille) this.vitesse +=x; },
9
10  repeindre(c) { this.couleur = c;}
11 };
12 velo.demarrer();
13 console.log(velo.bequille) ; // affiche "false"
14 velo.accelerer(20);
15 console.log(velo.vitesse) ; // affiche "20"
```

Constructeurs

- **Constructeur JavaScript** : fonction utilisant `this` pour initialiser le nouvel objet

```
1 let v = new Object(); // Équivalent à v = {}
2 v.couleur          = 'rouge';
3 v.nb_vitesses     = 7;
4 v.proprietaire    = {prenom : 'Alice', nom : 'Martin'};
5
6 // Mieux : constructeur
7 function Velo(c,n,p) {
8     this.couleur      = c;
9     this.nb_vitesses  = n;
10    this.proprietaire = p;
11 }
12 let v1 = new Velo('rouge', 7, {prenom : 'Alice', nom :
13     'Martin'});
14 let v2 = new Velo('bleu', 21, {prenom : 'Pierre', nom :
15     'Dupont'});
```

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux**
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements
- 8 Dessiner en JavaScript

Les tableaux en JavaScript

- **Tableau** : séquence d'éléments accessibles grâce à un indice entier

```
1 // Création :
2 let reals = ['Wong', 'Sciamma', 'Tarantino', 'Ducournau']
3
4 // Accès
5 console.log(reals[1]); // affiche "Sciamma"
6
7 // Modification
8 reals[2] = 'Hamaguchi';
9 console.log(reals); // affiche ["Wong", "Sciamma",
   "Hamaguchi", "Ducournau"]
10
11 // Propriétés
12 console.log(reals.length); // affiche "4"
13 reals.length = 3;
14 console.log(reals); // affiche ["Wong", "Sciamma",
   "Hamaguchi"]
15 reals.length = 4;
16 console.log(reals); // affiche ["Wong", "Sciamma",
   "Hamaguchi", <1 empty slot>"]
```

Absence de typage des tableaux

- On peut mettre toutes sortes de choses dans un tableau

```
1 let reals = ['Wong', 'Sciamma', 'Tarantino', 'Ducournau']
2
3 // Indice: 0 1 2 3 4 5 6
4 let tab = [1, 2, 3, {}, 'a', (x)=>x+2, reals];
5
6 console.log(tab[0]); // affiche "1"
7
8 tab[3].nom = 'Lee';
9 console.log(tab[3]); // affiche "Object {nom: "Lee"}"
10
11 let v = tab[5](8);
12 console.log(v); // affiche "10"
13
14 let l = tab[6][1];
15 console.log(l) ; // affiche "Sciamma"
```

L'objet Array

■ Déclaration et création

```
1 let nomDuTableau = new Array(taille); // taille =  
   nombre d'éléments du tableau
```

Utilisation : Les éléments sont indicés de 0 jusqu'à `taille - 1`

```
1 let tab = new Array(10);  
2 tab[3] = "Hou Hsiao-hsien";
```

■ Propriété `length` : nombre d'éléments du tableau

■ Quelques méthodes

- `concat` : `res=tab1.concat(tab2)` concatène `tab1` et `tab2` et met le résultat dans `res`
- `join` : `chaine=tab.join(car)` regroupe en une seule chaîne tous les éléments de `tab` séparés par le caractère séparateur `car` (par défaut `car = ', '`)
- `sort` : trie les éléments par ordre alphabétique
- `reverse` : inverse l'ordre des éléments
- `indexOf` : retourne le premier (plus petit) indice d'un élément égal à la valeur passée en paramètre à l'intérieur du tableau, ou `-1` si aucun n'a été trouvé

Plus de méthodes

■ `push` : ajoute un élément

```
1 let reals = ['Wong', 'Sciamma', 'Lee', 'Ducournau']
2 reals.push("Hou");
3 console.log(reals); // affiche ['Wong', 'Sciamma',
                        'Lee', 'Ducournau', 'Hou']
```

■ `pop` : retire le dernier élément

```
1 reals.pop();
2 console.log(reals); // affiche ['Wong', 'Sciamma',
                        'Lee', 'Ducournau']
```

■ `shift` : retire le premier élément

```
1 reals.shift();
2 console.log(reals); // affiche ['Sciamma', 'Lee',
                        'Ducournau']
```

■ `slice` : retourne un tableau avec les éléments entre deux indices

```
1 let qq_reals = reals.slice(1,3);
2 console.log(qq_reals); // affiche ['Lee', 'Ducournau']
```


Parcours de tableau : boucle `for`

■ Syntaxe classique

- Déjà vue dans de nombreux autres langages

```
1 let reals = ['Wong', 'Sciamma', 'Tarantino',  
2             'Ducournau']  
3 let i;  
4 for(i = 0; i < reals.length; i++) {  
5     console.log(reals[i]);  
6 }  
7 // Affiche :  
8 // Wong  
9 // Sciamma  
10 // Tarantino  
11 // Ducournau
```

...mais pas le plus habituel (ni le plus simple) en JavaScript

Parcours de tableau : syntaxe `for ... of`

```
1 let reals = ['Wong', 'Sciamma', 'Tarantino', 'Ducournau']  
2 for (let real of reals) {  
3   console.log(real);  
4 }
```

Parcours de tableau : syntaxe `forEach`

■ Meilleure syntaxe : `forEach`

- reçoit un *callback* et l'applique à chaque élément du tableau
- le *callback* reçoit en paramètre un élément du tableau


```
1 let reals = ['Wong', 'Sciamma', 'Tarantino', 'Ducournau']
2 reals.forEach( (real) => { console.log(real); });
3
4 let nombres = [1, 2, 4, 5, 7, 10, 13, 2004]
5 let somme = 0;
6 nombres.forEach( (nb) => { somme += nb; });
7 console.log(somme); // affiche 2046
```

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML**
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements
- 8 Dessiner en JavaScript

Les templates de chaînes dans une page Web (1/2)

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8" >
5     <title>Exemple : HTML + JavaScript</title>
6   </head>
7   <body>
8     <!-- Mauvaise pratique! -->
9     <script>
10      let nom      = 'Sciamma';
11      let prenom   = 'Céline';
12      let naissance = 1978;
13      let annee_courante = new Date().getFullYear();
14      let phrase = 'La réalisatrice ${prenom + ' ' + nom} a cette
15      année ${annee_courante - naissance} ans';
16      console.log(phrase); // affichage dans le terminal
17    </script>
18  </body>
</html>
```

 Exemple téléchargeable depuis Moodle

Les templates de chaînes dans une page Web (2/2)

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8" >
5     <title>Exemple : HTML + JavaScript</title>
6   </head>
7   <body>
8     <!-- Mauvaise pratique! -->
9     <script>
10      let nom      = 'Sciamma';
11      let prenom   = 'Céline';
12      let naissance = 1978;
13      let annee_courante = new Date().getFullYear();
14      let phrase = 'La réalisatrice ${prenom + ' ' + nom} a cette
15      année ${annee_courante - naissance} ans';
16      document.write(phrase); // affichage dans le corps de la
17      page Web
18    </script>
19  </body>
20 </html>
```



Exemple téléchargeable depuis Moodle

Utilisation d'une fonction JavaScript dans une page HTML

```
1 <!DOCTYPE html >
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8" >
5     <title>Exemple : HTML + JavaScript</title>
6     <!-- Mauvaise pratique! -->
7     <script>
8       function decubique(){
9         return Math.ceil(6*Math.random());
10      }
11    </script>
12  </head>
13  <body>
14    <h1>Tirage d'un dé à 6 faces</h1>
15    <script>
16      document.write('<p>Tirage = ' + decubique() + '</p>');
17    </script>
18  </body>
19 </html>
```

 Exemple téléchargeable depuis Moodle

Chargement de programmes JavaScript en HTML

Inclusion du code JavaScript dans la page HTML :

- balise `<script>`
- à placer dans la partie `<head>`
- ou (recommandé) : en fin de `<body>`
 - Avantage : permet de ne pas ralentir l'affichage de la page avec le téléchargement du JavaScript

```
1 <!DOCTYPE html >
2 <html lang="fr" >
3   <head >
4     ...
5   </head >
6   <body >
7     ....
8     <script src="monscript.js"></script >
9   </body >
10 </html >
```



Import d'une fonction JavaScript dans une page HTML

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8" >
5     <title>Exemple : HTML + JavaScript</title>
6     <script src="decubique.js"></script>
7   </head>
8   <body>
9     <h1> Tirage d'un dé à 6 faces</h1>
10    <!-- Pratique bof bof ! -->
11    <script>document.write('<p> Tirage = ' + decubique() + '</p>');</script>
12  </body>
13 </html>
```

Fichier decubique.js :

```
1 'use strict'
2 // Fonction retournant un tirage aléatoire entre 1 et 6
3 function decubique(){
4   return Math.ceil(6*Math.random());
5 }
```

Démonstration

 Exemple téléchargeable depuis Moodle

Bonne pratique

✓ Séparation du JavaScript

Le code JavaScript devrait être (quasi) intégralement séparé du HTML.

Dans l'idéal, la seule modification du code HTML est l'insertion du script grâce à la balise `<script>`.

Outline

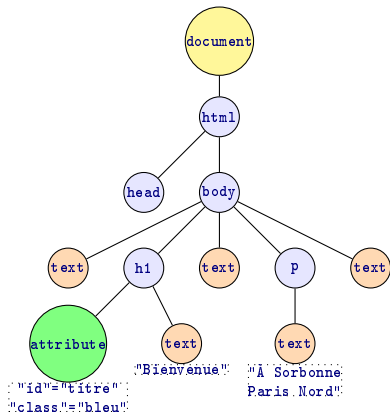
- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript**
- 7 Les évènements
- 8 Dessiner en JavaScript

JavaScript et le DOM

- Tous les navigateurs disposent d'un interprète JavaScript embarqué
- Interaction de JavaScript avec le document : **DOM**
 - Rappel : DOM = *Document Object Model*
- Réaction suite à des actions de l'utilisateur ou utilisatrice : **événements**

DOM : rappel

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head> ... </head>
4   <body>
5     <h1 id="titre"
6       class="bleu">Bienvenue</h1>
7     <p>À Sorbonne Paris Nord</p>
8   </body>
</html>
```



Manipuler le DOM avec JavaScript

- JavaScript fournit une interface pour manipuler le DOM :
 - sélectionner et accéder à des éléments de l'arbre DOM, naviguer dans le DOM
 - créer, insérer, supprimer, modifier des éléments dans le DOM
 - écouter des évènements produits par des éléments dans le DOM

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript**
 - **Sélection d'éléments du DOM**
 - Déplacement dans le DOM
 - Modification du DOM
- 7 Les évènements

Sélectionner des éléments du DOM

- `document.getElementById("id")`
Retourne le nœud (unique) dont l'identifiant est `id`
- `document.getElementsByTagName("mabalise")`
Retourne les nœuds dont la balise est `mabalise`
- `document.getElementsByClassName("maclasse")`
Retourne les nœuds dont la classe est `maclasse`

Sélectionner des éléments du DOM : exemple

```
1 <html lang="fr">
2   <head> ... </head>
3   <body>
4     <section class="main">
5       <p id="p1">paragraphe 1</p>
6       <p id="p2">paragraphe 2</p>
7       <p id="p3">paragraphe 3</p>
8     </section>
9   </body></html>
```

```
1 let premier = document.getElementById("p1");
2 console.log(premier.nodeName, premier.id); // affiche "P p1"
3
4 let paragraphes = document.getElementsByTagName("p");
5 console.log(paragraphes[1].nodeName, paragraphes[1].id); //
  affiche "P p2"
6
7 let mains = document.getElementsByClassName("main");
8 console.log(mains[0].nodeName, mains[0].className); //
  affiche "SECTION main"
```

Sélectionner des éléments du DOM : requêtes

- `element.querySelector(selecteur)`

Retourne parmi les descendants du nœud sur lequel on l'invoque le 1^{er} élément qui correspond au `selecteur`

- `element` : peut être `document` ou un nœud quelconque
- `selecteur` : sélecteur CSS

- `element.querySelectorAll(selecteur)`

Retourne parmi les descendants du nœud sur lequel on l'invoque la liste contenant tous les éléments qui correspondent au `selecteur`

Sélectionner des éléments du DOM par requête : exemple

```
1 <html lang="fr">
2   <head> ... </head>
3   <body>
4     <section class="main">
5       <p id="p1">paragraphe 1</p>
6       <p id="p2">paragraphe 2</p>
7       <p id="p3">paragraphe 3</p>
8     </section>
9   </body></html>
```

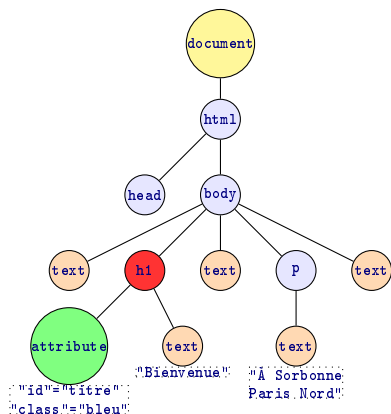
```
1 let premier = document.querySelector("#p1");
2 console.log(premier.nodeName, premier.id); // affiche "P p1"
3
4 let paragraphes = document.querySelectorAll("p");
5 console.log(paragraphes[1].nodeName, paragraphes[1].id); //
  affiche "P p2"
6
7 let mains = document.querySelectorAll(".main");
8 console.log(mains[0].nodeName, mains[0].className); //
  affiche "SECTION main"
```

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript**
 - Sélection d'éléments du DOM
 - Déplacement dans le DOM**
 - Modification du DOM
- 7 Les évènements

Se déplacer dans le DOM : nom du nœud courant

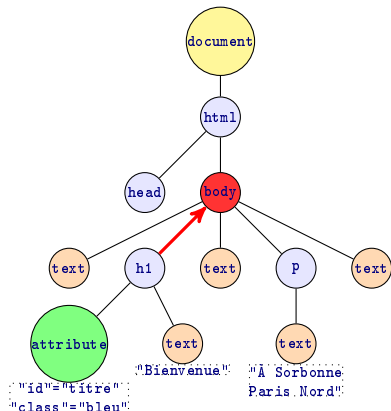
■ Nom du nœud courant : `nodeName`



```
1 let noeud =  
    document.querySelector("#titre")  
2 console.log(noeud.nodeName); //  
    affiche "H1"
```

Se déplacer dans le DOM : nœud parent

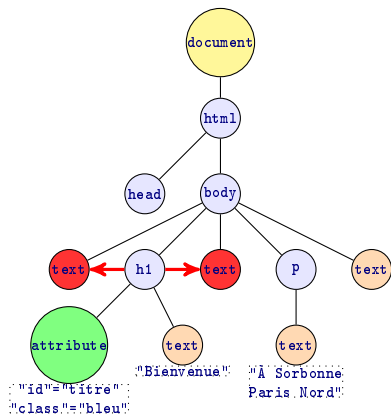
■ Accès au nœud parent : `parentNode`



```
1 let noeud =  
    document.querySelector("#titre")  
2 let parent = noeud.parentNode;  
3 console.log(parent.nodeName); //  
    affiche "BODY"
```

Se déplacer dans le DOM : nœuds frères

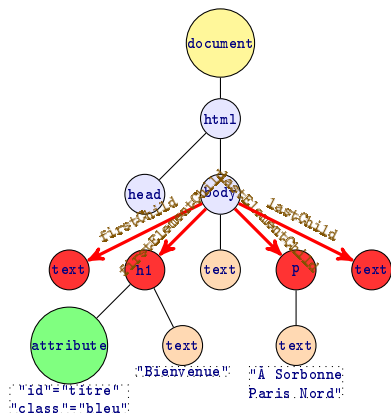
- Accès au nœud frère antérieur : `previousSibling`
- Accès au nœud frère postérieur : `nextSibling`
- *siblings* : nœuds enfants du parent du nœud courant



```
1 let noeud =
    document.querySelector("#titre")
2 let frere_precedent =
    noeud.previousSibling;
3 let frere_suivant =
    noeud.nextSibling;
4 console.log(frere_precedent.nodeName)
    // affiche "#text"
5 console.log(frere_suivant.nodeName)
    // affiche "#text"
```

Se déplacer dans le DOM : nœuds enfants (1/2)

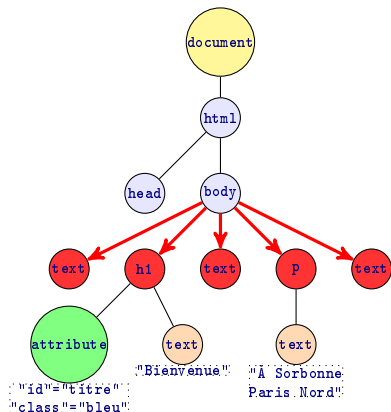
- Accès au premier enfant : `firstChild`
- Accès au premier enfant qui est un élément : `firstElementChild`
- Accès au dernier enfant : `lastChild`
- Accès au dernier enfant : `lastElementChild`



```
1 let noeud =
    document.querySelector("body");
2 let fs = noeud.firstChild ;
3 console.log(fs.nodeName); //
    affiche "#text"
4 console.log(noeud.firstElementChild
5 .nodeName); // affiche "H1"
6 console.log(noeud.firstElementChild
7 .firstChild.textContent); //
    affiche "Bienvenue"
```


Se déplacer dans le DOM : nœuds enfants (2/2)

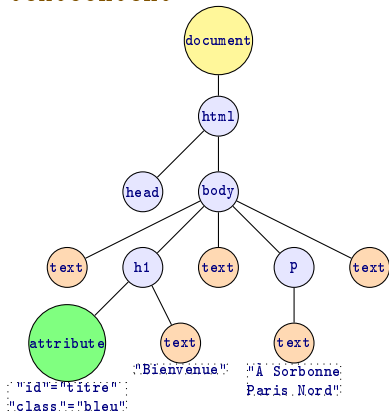
- Liste de tous les nœuds enfants : `childNodes`
- Nombre de nœuds enfants qui sont des éléments : `childElementCount`



```
1 let noeud =
    document.querySelector("body");
2 let enfants = noeud.childNodes;
3 console.log(enfants[3].nodeName);
    // affiche "P"
4 console.log(enfants.length); //
    affiche 5
5 console.log(noeud.childElementCount);
    // affiche 2
```

Valeur texte d'un nœud

La valeur textuelle d'un nœud de type « texte » peut être récupérée par l'attribut `textContent`



```
1 let noeud =  
    document.querySelector("body");  
2  
3 console.log(noeud.firstChild  
4 .firstChild.textContent); //  
    affiche "Bienvenue"
```

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript**
 - Sélection d'éléments du DOM
 - Déplacement dans le DOM
 - Modification du DOM**
- 7 Les évènements

Modifier les attributs CSS du document

- Modification de l'apparence d'un document : en ajoutant ou en retirant des classes CSS sur un élément
 - `elt.classList` : liste des classes portées par `elt` (propriété non modifiable directement)
 - Ajout : `elt.classList.add("nouvelle")` : ajoute de la classe "nouvelle" à l'élément `elt`
 - Suppression : `elt.classList.remove("suppr")` : supprime la classe "suppr" de l'élément `elt`
 - Inversion : `elt.classList.toggle("classe")` : ajoute/supprime la classe "classe" de l'élément `elt`
 - Test : `elt.classList.contains("classe")` : rend `vrai` si l'élément `elt` porte la classe "classe"

Modifier le document

■ Méthodologie

- 1 créer des nœuds, puis
- 2 les insérer dans l'arbre, puis
- 3 supprimer des nœuds de l'arbre (le cas échéant)

Création de nœuds

- **createElement** : création d'un nœud de type élément

```
1 let b = document.createElement("b"); // création d'une  
   balise "<b>" (texte gras)
```

- **createTextNode** : création d'un nœud de type texte

```
1 let t = document.createTextNode('Bienvenue');
```

- **setAttribute** : affectation à un nœud d'un attribut avec sa valeur

```
1 b.setAttribute("id", "tresgras");
```

Insertion de nœuds dans le DOM

- **appendChild**: insertion d'un nœud comme dernier enfant

```
1 noeud.appendChild(n); // insère le noeud n comme  
dernier enfant de "noeud"
```

- **insertBefore**: insertion d'un nœud avant le nœud passé en paramètre

```
1 noeud.insertBefore(n, nb); // insère le noeud n avant  
nb dans les enfants de "noeud"
```

- **replaceChild**: remplace un nœud

```
1 noeud.replaceChild(n, nb); // remplace nb par n dans  
les enfants de "noeud"
```

- **removeChild**: supprime un nœud des enfants

```
1 noeud.removeChild(n); // supprime n de la liste des  
enfants de "noeud"
```

Création et insertion de nœuds : exemple

```
1 window.addEventListener("load", () => {
2     document.addEventListener("keypress", (e) => {
3         console.log("key: ", e.key, "keycode: ", e.keyCode);
4         if(e.key === 'm'){
5             let body = document.querySelector("body");
6             let nouveaup = document.createElement("P");
7             nouveaup.setAttribute("id", "p4");
8             let t = document.createTextNode("Et à Villetaneuse");
9             nouveaup.appendChild(t);
10            body.appendChild(nouveaup);
11            // insérer 1 ligne après le h1
12            let p = document.querySelector("body>p");
13            body.insertBefore(document.createElement("HR"), p);
14        } // end if
15    }); // end listener
16 });
```



Démonstration

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements**
- 8 Dessiner en JavaScript

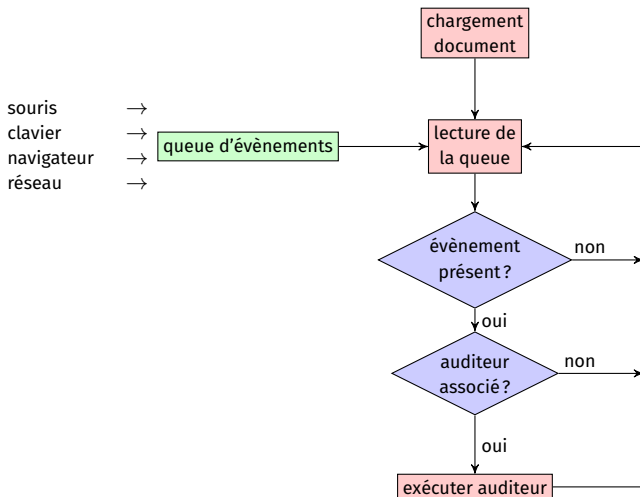
Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements**
 - Les évènements en JavaScript
 - Les gestionnaires d'évènements

Les évènements en JavaScript

- Évènement : **action de l'utilisateur** sur l'interface (ou du réseau, ou ...)
- Pour réagir à ces actions : on lie une fonction à un évènement
 - *handler* (gestionnaire d'évènements)

La boucle d'évènements JavaScript



⚠ Un seul auditeur est exécuté à la fois!

Les évènements et le DOM

- Les objets du DOM génèrent des évènements correspondant souvent à des actions de l'utilisateur sur l'interface (ils sont **observables**)
- Une application JavaScript dans le navigateur est une application **conduite par des évènements** (*event driven*)
- La boucle d'évènements est réalisée par l'interprète JavaScript

Quelques évènements du DOM en JavaScript

Évènement	Type de nœud	Action détectée
<code>load</code>	<code>document, frameset</code>	Chargement complet du DOM
<code>click</code>	<code>element</code>	Clic de souris
<code>mousedown</code>	<code>element</code>	Bouton de souris enfoncé
<code>mouseup</code>	<code>element</code>	Bouton de souris relâché
<code>mouseover</code>	<code>element</code>	Passage du pointeur au-dessus
<code>mouseout</code>	<code>element</code>	Sortie du pointeur
<code>keypress</code>	<code>element</code>	Touche de clavier pressée
<code>blur</code>	<code>element</code>	Perte du focus (utile dans un formulaire)
<code>focus</code>	<code>element</code>	Acquisition du focus
<code>change</code>	<code>element</code>	Modification de la valeur
<code>resize</code>	<code>document, element</code>	Changement de taille
<code>scroll</code>	<code>document, element</code>	Utilisation de la molette

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements**
 - Les évènements en JavaScript
 - Les gestionnaires d'évènements**

Enregistrer le gestionnaire d'évènement sur un nœud

- **Trois méthodes principales** pour enregistrer la fonction gestionnaire (*handler*) sur un nœud pour un évènement
 - 1 Via la propriété `on<event>` dans le HTML (⚠ mauvaise pratique!)
 - 2 Via la propriété `on<event>` des nœuds
 - 3 Via un auditeur dédié (souvent la **meilleure pratique**)

- Quand un évènement `e` survient : `e.target` contient la cible de l'évènement (propriété de `Event`)
 - Documentation : <https://developer.mozilla.org/fr/docs/Web/API/Event>

Méthode 1 : propriété `on<event>` du HTML

- Associer un appel JavaScript à un évènement directement dans le HTML

```
1 <!DOCTYPE html >
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8">
5     <title>Gestionnaire d'évènement méthode 2</title>
6   </head>
7   <body>
8     <p onclick="alert('Sorbonne !');">Sorbonne Paris
9     Nord</p>
10    <p onmouseover="console.log('Galilée');">Institut
11    Galilée</p>
12  </body>
13 </html>
```

Méthode 1 : propriété `on<event>` du HTML

- Associer un appel JavaScript à un évènement directement dans le HTML

```
1 <!DOCTYPE html >
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8">
5     <title>Gestionnaire d'évènement méthode 2</title>
6   </head>
7   <body>
8     <p onclick="alert('Sorbonne !');">Sorbonne Paris
9     Nord</p>
10    <p onmouseover="console.log('Galilée');">Institut
11    Galilée</p>
12  </body>
13 </html>
```



Mauvaise pratique

Cette pratique est une mauvaise pratique (dépréciée), et ne devrait pas être utilisée

Méthode 2 : propriété `on<event>` des nœuds

- Donner une valeur à la propriété `on<evt>` d'un nœud du DOM
 - `load` ⇒ `onload`
 - `click` ⇒ `onclick`
 - etc.
- Lorsque l'évènement se produit, cette valeur est utilisée comme gestionnaire d'évènement

```
1 node.onclick = () => { ... };  
2 node.onmouseover = animateButton;  
3 window.onload = () => { ... };
```



Un seul gestionnaire

Cette méthode ne permet pas d'enregistrer plusieurs gestionnaires (*handlers*) sur le même évènement

Méthode 2 : exemple 1

```
1 <!DOCTYPE html >
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8">
5     <title>Gestionnaire d'évènement méthode 2</title>
6     <script>
7       document.onclick = () => {console.log("Clic dans la
8 page détecté !")};
9       document.onkeypress = affToucheClavier;
10      function affToucheClavier(e){
11        console.log("Touche pressée : ", e.key, "Code :
12 ", e.keyCode);
13      }
14    </script>
15  </head>
16  <body>
17    <h1>Évènements clavier et clic</h1>
18  </body>
19 </html >
```

Méthode 2 : exemple 2

```
1 <!DOCTYPE html >
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8">
5     <title>Gestionnaire d'évènement méthode 2</title>
6   </head>
7   <body>
8     <p id="SPN">Sorbonne Paris Nord</p>
9     <p id="IG">Institut Galilée</p>
10    <script>
11      let spn = document.getElementById("SPN");
12      let ig = document.querySelector("#IG");
13
14      spn.onclick = (event) => {alert("Sorbonne !");};
15      ig.onmouseover = (event) => {alert ("Galilée !");};
16    </script>
17  </body>
18 </html >
```



Démonstration

Méthode 3 : auditeur

- Enregistrement d'un auditeur (ou *listener* ou écouteur) sur un nœud pour un évènement

```
1 node.addEventListener("evt", handler);  
2 // "evt"      : l'évènement concerné  
3 // handler   : une fonction recevant un objet event en  
   paramètre
```

- Il est possible d'enregistrer plusieurs auditeurs sur le même nœud pour un même évènement

```
1 lien.addEventListener("click", (event) => {alert("clie  
  !");});  
2 lien.addEventListener("click", (event) =>  
  {console.log("clie !");});
```

Méthode 3 : exemple de déclaration

```
1 <html lang="fr">
2   <head> ... </head>
3   <body>
4     <p id="SPN">Sorbonne Paris Nord</p>
5     <p id="IG">Institut Galilée</p>
6     <script src="handler-methode3.js"></script>
7   </body>
8 </html>
```

```
1 let spn = document.getElementById("SPN");
2 let ig  = document.querySelector("#IG");
3
4 spn.addEventListener("click", (event) => {alert("Sorbonne
5   !");});
6 spn.addEventListener("click", (event) =>
7   {console.log("Sorbonne !");});
8 ig.addEventListener("mouseover", (event) =>
9   {console.log("Galilée !");});
```



Démonstration

Méthode 3 : chargement du DOM

⚠ Chargement du DOM

Il est important de s'assurer que le DOM est entièrement chargé avant de sélectionner des nœuds et de déclarer des auditeurs : insérer les déclarations d'auditeurs dans... un auditeur sur l'évènement "load"

```
1 'use strict'
2 window.addEventListener("load", () => {
3   let spn = document.getElementById("SPN");
4   let ig  = document.querySelector("#IG");
5
6   spn.addEventListener("click", (event) => {alert("Sorbonne
7     !");});
8   spn.addEventListener("click", (event) =>
9     {console.log("Sorbonne !");});
10  ig.addEventListener("mouseover", (event) =>
11    {console.log("Galilée !");});
12 });
```


Propriétés et méthodes des évènements

- Un auditeur reçoit comme seul argument un objet `event`
- Selon le type de l'évènement, on a accès à différentes propriétés/méthodes :
 - pour accéder à la `cible` (nœud du DOM ayant reçu l'évènement) :
 - `event.target` : dans tous les cas
 - `this` : dans les auditeurs définis avec `function()`
 - pour annuler l'évènement : `event.preventDefault()`

Auditeur : exemple plus élaboré

```
1 'use strict'
2 window.addEventListener("load", () => {
3     document.querySelector("#IG").addEventListener("click", (e) =>
4         {e.target.classList.toggle("vert");});
5
6     document.querySelector("h1").addEventListener("mouseover", (e)
7         => {console.log("Souris passée sur le titre");
8             document.querySelectorAll(".discipline").forEach( (d) =>
9                 {d.classList.toggle("bleu");}); });
10
11     document.addEventListener("mousedown", (e) => {
12         console.log("Clic @ x:", e.clientX, "y:", e.clientY); });
13     document.addEventListener("keypress", (e) => {
14         console.log("key: ", e.key, "keycode: ", e.keyCode);
15         switch (e.key) {
16             case 'b': document.body.style.backgroundColor = 'white';
17             break
18             case 'o': document.body.style.backgroundColor = 'orange';
19             break
20         }
21     });
22 });
```



Démonstration

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements
- 8 Dessiner en JavaScript**

Dessiner dans une page HTML

■ Utilisation de l'élément HTML `canvas`

- se comporte comme une image

```
1 <canvas id="canevas" width="300" height="300">
2   Désolé, votre navigateur ne prend pas en charge canvas
3 </canvas>
```

■ Sur un objet associé à `canvas` :

- Méthode `getContext` donne accès au contexte graphique du canevas et permet ensuite de dessiner
- Zone de dessin : origine $(0, 0)$ en haut à gauche du canevas

Canevas : exemple basique (HTML)

```
1 <!DOCTYPE html >
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8">
5     <title>Exemple de canevas basique</title>
6   </head>
7   <body>
8     <h1>Canevas</h1>
9     <canvas id="canevas" width="300" height="300">
10      Désolé, votre navigateur ne prend pas en charge «
11      &nbsp;canvas&nbsp;».
12     </canvas>
13     <script src="canevas-basique.js"></script>
14 </body>
</html>
```

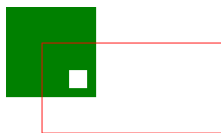
Canevas : exemple basique (JavaScript)

```
1 let canvas = document.getElementById('canevas');
2 let contexte = canvas.getContext('2d');
3 contexte.fillStyle = 'green';
4 contexte.fillRect(10, 10, 100, 100);
5 contexte.strokeStyle = 'red';
6 contexte.strokeRect(50, 50, 200, 100);
7 contexte.clearRect(80, 80, 20, 20);
```


Canevas : exemple basique (JavaScript)

```
1 let canvas = document.getElementById('canevas');
2 let contexte = canvas.getContext('2d');
3 contexte.fillStyle = 'green';
4 contexte.fillRect(10, 10, 100, 100);
5 contexte.strokeStyle = 'red';
6 contexte.strokeRect(50, 50, 200, 100);
7 contexte.clearRect(80, 80, 20, 20);
```

Canevas



 Démonstration

 Exemple téléchargeable depuis Moodle

Formes géométriques : rectangles

- **fillRect** : Dessine un rectangle rempli

```
1 fillRect(x, y, largeur, hauteur)
```

- **strokeRect** : Dessine un contour rectangulaire

```
1 strokeRect(x, y, largeur, hauteur)
```

- **clearRect** : Efface la zone rectangulaire spécifiée, la rendant complètement transparente

```
1 clearRect(x, y, largeur, hauteur)
```


Formes géométriques : trajets, lignes, arcs

Quatre étapes

- 1 création du trajet : `beginPath()`
- 2 séquence d'instructions de dessin pour dessiner sur le trajet (lignes, arcs...)
 - `moveTo(x,y)` : déplacement en (x,y) sans trace
 - `lineTo(x,y)` : trace une ligne depuis la position courante jusqu'au point (x,y)
 - `arc(x, y, rayon, angleInitial, angleFinal, antihoraire)` : dessine un arc de cercle
- 3 fermer le trajet : `closePath()`
- 4 le tracer (`fill()`) ou le remplir (`stroke()`) pour le faire apparaître

Formes géométriques : arcs et cercles

- **Arc de cercle** : `arc(x, y, rayon, angleInitial, angleFinal, antihoraire)`
 - dessine un arc de cercle centré sur la position `(x, y)`, de rayon `r`, commençant à `angleInitial` et finissant à `angleFinal` en allant dans le sens antihoraire ssi `antihoraire === true` (par défaut : horaire)
- **Cercle** : utiliser `arc` avec `angleInitial = 0`, `angleFinal = Math.PI * 2`

```
1 // Dessine un disque rouge plein de centre (75,75) et
   de rayon 50
2 beginPath();
3 contexte.arc(75, 75, 50, 0, Math.PI * 2, true);
4 closePath();
5 contexte.fillStyle = "red";
6 fill();
```

Formes géométriques : exemple (1/3)

```
1 function dessiner() {
2   let canevas = document.getElementById('canevas');
3   if (canevas.getContext) {
4     let contexte = canevas.getContext('2d');
5
6     // Triangle plein
7     contexte.beginPath();
8     contexte.moveTo(25, 25);
9     contexte.lineTo(105, 25);
10    contexte.lineTo(25, 105);
11    contexte.closePath();
12    contexte.fillStyle = 'blue';
13    contexte.fill();
14    // Triangle filaire
15    contexte.beginPath();
16    contexte.moveTo(125, 125);
17    contexte.lineTo(125, 45);
18    contexte.lineTo(45, 125);
19    contexte.closePath();
20    contexte.strokeStyle = 'red';
21    contexte.stroke();
```

Formes géométriques : exemple (2/3)


```
1 // Cercle
2 contexte.beginPath();
3 contexte.arc(150, 50, 20, 0, 2*Math.PI, 0);
4 contexte.closePath();
5 contexte.strokeStyle = 'green';
6 contexte.stroke();
7
8 // Demi-disque
9 contexte.beginPath();
10 contexte.arc(150, 100, 20, Math.PI/4, 5/4*Math.PI, 0);
11 contexte.closePath();
12 contexte.fillStyle = 'orange';
13 contexte.fill();
14 }
15 }
16
17 window.addEventListener('load', function() {
18     dessiner();
19 });
```

Formes géométriques : exemple (3/3)

Rendu :

Canevas



 Exemple téléchargeable depuis Moodle

Les couleurs

- Deux propriétés importantes du contexte de canevas
 - `fillStyle(couleur)` : définit la couleur utilisée lors du remplissage
 - `strokeStyle(couleur)` : définit la couleur utilisée lors du tracé
 - ...où `couleur` est une chaîne représentant une couleur CSS, un objet gradient ou un objet motif

- Définition des couleurs : nom, valeur hexadécimale, RGB, RGBA

```
1 // Définitions possibles de la couleur orange
2 contexte.fillStyle = "orange";
3 contexte.fillStyle = "#FFA500";
4 contexte.fillStyle = rgb(255,165,0);
5 contexte.fillStyle = rgba(255,165,0,1);
```

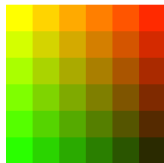
- Documentation sur les couleurs :

<https://developer.mozilla.org/fr/docs/Web/API/Event>

Les couleurs : exemple

```
1 function dessiner() {
2   let canevas =
3     document.getElementById('canevas');
4   if (canevas.getContext) {
5     let contexte = canevas.getContext('2d');
6
7     for (let i = 0; i < 6; i++) {
8       for (let j = 0; j < 6; j++) {
9         contexte.fillStyle = 'rgb(' +
10          Math.floor(255 - 42.5 * i) + ',' +
11          Math.floor(255 - 42.5 * j) + ',0)';
12         contexte.fillRect(j * 25, i * 25, 25, 25);
13       }
14     }
15 }
16
17 window.addEventListener('load', function() {
18   dessiner();
19 });
```

Canevas



 Exemple téléchargeable depuis Moodle

Outline

- 1 Définition et syntaxe de base
- 2 Les fonctions
- 3 Les objets
- 4 Les tableaux
- 5 Import de JavaScript dans du HTML
- 6 Manipulation du DOM avec JavaScript
- 7 Les évènements
- 8 Dessiner en JavaScript

Formulaires et JavaScript : l'objet `form`

```
1 <form id="form1" name="monForm"  
   action="mailto:toto@uspn.fr" method="post "  
   enctype="text/plain">  
2   Saisie du texte&nbsp;: <input type="text" name="saisie">  
3     <input type="submit" value="Valider le formulaire">  
4 </form>
```

Plusieurs manières d'accéder à l'objet JavaScript `form`:

```
1 let f1 = document.getElementById("form1"); //recherche avec  
   id (MÉTHODE RECOMMANDÉE)  
2 let f2 = document.forms[0] ; // indice 0 du tableau forms  
3 let f3 = document.forms['monForm'] ; // indice name  
4 let f4 = document.monForm ; //accès direct avec name
```

 Exemple téléchargeable depuis Moodle

Quelques propriétés de l'objet `form`

- `name` : nom du formulaire
- `action` : `mailto` ou adresse de la page vers laquelle est envoyé le formulaire lors de sa soumission (par appui sur le bouton HTML `submit` ou l'appel de la méthode JavaScript `submit()`)
- `method` : méthode de soumission : `post` ou `get`
- `enctype` : type d'encodage du contenu, par ex `'plain/text'`
- `elements` : tableau contenant tous les objets correspondants aux composants du formulaire

Quelques méthodes de l'objet `form`

- `reset()` : réinitialise le formulaire
- `submit()` : soumet le formulaire

Quelques propriétés des composants des formulaires

Certaines de ces propriétés sont communes à tous les composants; d'autres sont spécifiques à certains composants

- `type` : type de l'élément ("`text`", "`button`", "`radio`", "`checkbox`",...)
- `name` : nom de l'élément (rappel HTML : même `name` pour des boutons radio groupés)
- `value` : contenu du champ pour une zone de saisie
- `id` : identifiant de l'élément
- `disabled` : booléen à `true` si le champ de saisie est grisé et non accessible
- `checked` : booléen à `true` si une case à cocher ou un bouton radio est coché
- `selectedIndex` : indice de l'élément sélectionné dans une liste

Quelques méthodes des composants des formulaires

- `blur()` : fait perdre le focus à un composant
- `focus()` : met le focus sur un composant
- `change()` : déclenche un changement dans un composant
- `click()` : déclenche un clic de souris sur le composant
- `select()` : sélectionne le contenu d'un élément de formulaire

Exemple : boîtes à cocher (HTML)

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>...</head>
4   <body>
5     <h1>Quiz cinéma</h1>
6     <p>Parmi ces noms, qui a déjà gagné la Palme d'or&nbsp;?</p>
7     <form id="formulaire">
8       <p><input type="checkbox" id="besson"><label>Luc
9       Besson</label></p>
10      <p><input type="checkbox" id="ducournau"><label>Julia
11      Ducournau</label></p>
12      <p><input type="checkbox" id="imamura"><label>Shōhei
13      Imamura</label></p>
14      <p><input type="checkbox" id="tarantino"><label>Quentin
15      Tarantino</label></p>
16      <p><input type="checkbox" id="vansant"><label>Gus Van
17      Sant</label></p>
18      <p><button id="verifier">Vérifier</button></p>
19      <p><button id="reinitialiser">Réinitialiser</button></p>
20    </form>
21    <script src="formulaire-checkbox.js"></script>
22  </body>
23 </html>
```

Exemple : boîtes à cocher (JavaScript)

```
1 function avertissement(e) {
2   if(e.target.checked === true){alert("N'importe quoi !");}}
3 function verification(e) {
4   let c1 = document.querySelector("#besson");
5   let c2 = document.querySelector("#ducournau");
6   let c3 = document.querySelector("#imamura");
7   let c4 = document.querySelector("#tarantino");
8   let c5 = document.querySelector("#vansant");
9   if (c1.checked === false && c2.checked === true && c3.checked
10      === true && c4.checked === true && c5.checked === true)
11     alert("Bonne réponse !");
12   else
13     alert("Faux, retournez au cinéma !");}
14 document.querySelector("#verifier").addEventListener('click',
15   verification);
16 document.querySelector("#reinitialiser").addEventListener('click',
17   (e) => {document.getElementById('formulaire').reset();});
18 document.querySelector("#besson").addEventListener('click',
19   avertissement);
```

Exemple : boîtes à cocher (rendu)

Quiz cinéma

Parmi ces noms, qui a déjà gagné la Palme d'or ?

- Luc Besson
- Julia Ducournau
- Shōhei Imamura
- Quentin Tarantino
- Gus Van Sant

Vérifier

Réinitialiser



Démonstration



Exemple téléchargeable depuis Moodle

Exemple : boutons radio et zone de saisie (HTML)

```
1 <h1>Conversion de caractères</h1>
2
3 <form id="formulaire">
4   <p>Nom&nbsp;: <input type="text" name="nom" size=20
5     ></p>
6   <p>Prénom&nbsp;: <input type="text" name="prenom" size=20
7     ></p>
8   <p>Ville&nbsp;: <input type="text" id="ville"
9     name="ville" size=20 ></p>
10  <p>Convertir en</p>
11  <p><input type="radio" name="conversion"
12    id="maj">Majuscules</p>
13  <p><input type="radio" name="conversion"
14    id="min">Minuscules</p>
15  <p><input type="radio" name="conversion"
16    id="aucune">Aucune</p>
17  <p><button id="reinitialiser">Réinitialiser</button></p>
18 </form>
19 <script src="formulaire-radio.js"></script>
```

Exemple : boutons radio et zone de saisie (JavaScript)

```
1 function convertirChamp(idChamp) {
2   if (document.querySelector("#maj").checked)
3     idChamp.value = idChamp.value.toUpperCase();
4   else if (document.querySelector("#min").checked)
5     idChamp.value = idChamp.value.toLowerCase();
6 }
7 function convertirTout(e) {
8   let formulaire = document.getElementById('formulaire');
9   convertirChamp(formulaire.nom);
10  convertirChamp(formulaire.prenom);
11  convertirChamp(formulaire.ville);
12 }
13 window.addEventListener('load', function() {
14   document.querySelector("#maj").addEventListener('click',
15     convertirTout);
16   document.querySelector("#min").addEventListener('click',
17     convertirTout);
18   document.querySelector("#ville").addEventListener('change', (e)
19     => convertirChamp(e.target));
20
21   document.querySelector("#reinitialiser").addEventListener('click',
22     (e) => {document.getElementById('formulaire').reset();});
23 });
```

Exemple : boutons radio et zone de saisie (rendu)

Conversion de caractères

Nom :

Prénom :

Ville :

Convertir en

Majuscules

Minuscules

Aucune

Démonstration

 Exemple téléchargeable depuis Moodle

Exemple : liste de sélection (HTML)

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8">
5     <title>Exemple de formulaire avec sélection</title>
6   </head>
7   <body>
8     <h1>Sélection</h1>
9
10    <form id="formulaire">
11      <label for="liste">Choisir votre ville
12      préférée&nbsp;:</label>
13      <select id="liste" name="liste">
14        <option value="nancy">Ville de Nancy</option>
15        <option value="paris">Ville de Paris</option>
16        <option value="villetaneuse">Ville de
17        Villetaneuse</option>
18      </select>
19      <input type="text" readonly size=40 id="resultat"
20      value="Vous avez sélectionné :">
21    </form>
22    <script src="formulaire-select.js"></script>
23  </body>
24 </html>
```

Exemple : liste de sélection (JavaScript)

```
1 'use strict'
2
3 function modifier(e){
4     let el = e.target;
5     document.getElementById("resultat").value =
6         document.getElementById("resultat").value + " " +
7         el.options[el.selectedIndex].value;
8 }
9
10 window.addEventListener('load', function() {
11     document.querySelector("#liste").addEventListener('change',
12         modifier);
13 });
```

Exemple : liste de sélection (rendu)

Sélection

Choisir votre ville préférée :

Ville de Nancy	▼	Vous avez sélectionné :
Ville de Nancy		
Ville de Paris		
Ville de Villetaneuse		

Démonstration

 Exemple téléchargeable depuis Moodle

QCM noté en TP

QCM notés

Un ou plusieurs QCM noté est susceptible d'être organisé pendant les séances de TP sans préavis.

Sources et références

Sources et références

Références

- **Frédéric Delobel** : JavaScript et jQuery – La programmation Web par la pratique
- **Robin Nixon** : Développer un site web en PHP, MySQL JavaScript jQuery, CSS3 et HTML5 : Un guide étape par étape pour créer des sites Web dynamiques (O'Reilly)


Sources

- Ce cours est largement inspiré des supports de cours de Slim OUNI, Isabelle DEBLED-RENESSON et Étienne ANDRÉ (Université de Lorraine)


Crédits

Source des images utilisées I




Titre : warning sign
Auteur : ?
Source : https://commons.wikimedia.org/wiki/File:Warning_sign.png
Licence : 



Titre : Check Mark CSS Green
Auteur : Pigeon43
Source : https://commons.wikimedia.org/wiki/File:Check_Mark_CSS_Green.svg
Licence : 



Titre : small-n-flat
Auteur : paomedia
Source : <https://commons.wikimedia.org/wiki/File:Device-computer-icon.png>
Licence : 




Titre : Nuvola filesystems folder download
Auteur : David Vignoni
Source : https://commons.wikimedia.org/wiki/File:Nuvola_filesystems_folder_download.png
Licence : 




Titre : prompt
Auteur : Étienne André

Source des images utilisées II

Source :
Licence : 


Canevas



Titre : canevas basique
Auteur : Étienne André
Source :
Licence : 


Canevas



Titre : canevas et formes
Auteur : Étienne André
Source :
Licence : 


Canevas



Titre : canevas et couleurs
Auteur : Étienne André
Source :
Licence : 

Quiz cinéma


Formez vos équipes, qui a déjà regardé le Palace d'Or ?
 Est-ce possible
 Quelle destination
 Pourquoi les hommes
 Pourquoi les femmes
 Est-ce que vous savez
 Pourquoi
 Pourquoi

Titre : Formulaire
Auteur : Étienne André
Source :
Licence : 

Source des images utilisées III


Conversion de caractères



Titre : Formulaire
Auteur : Étienne André
Source :
Licence : 

Selection



Titre : Formulaire
Auteur : Étienne André
Source :
Licence : 

Licence de ce document

Ce support de cours peut être réutilisé, modifié et republié selon les termes de la licence Creative Commons **Attribution-NonCommercial-ShareAlike 4.0 Unported (CC BY-NC-SA 4.0)**



<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Auteur : Étienne André

(Source \LaTeX disponible aux enseignant·e·s sur demande)

UNIVERSITÉ
SORBONNE
PARIS NORD

Institut
Galilée

Version : 8 février 2023